Chapter 3

Data model

3.1 Overview

Modeling plant operations is the very first step on the solution of a scheduling problem. This is a hard work when dealing with complex recipes that do not fit into standard models (flow-shop, job-shop...) and objectives (makespan, tardiness), that require simultaneous combination of different policies (NIS,ZW,UIS, FIS...) and co-ordination of specific activities.

The modeling framework introduced in this thesis is based on previous works of Graells et al. (1995, 1996) where production lines were first split into subtrains and next mini-jobs of the resulting intermediate products were sequenced under the corresponding storage and precedence constraints. Yet, this approach has been improved and extended so that branched production processes sharing intermediates may be considered. This modeling framework also follows the guidelines of the ISA SP88 standard (International Soc. for Measurement and Control, 1995, 1999), so the resulting model is consistent with the models used in process control. The structure of processes (individual tasks, whole subtrains or complex structures of activities) and related materials (raw, intermediate or final products) are characterized by means of a Processing Network, which describes the material balance. Later on, the structure of the activities to be performed within each process is represented through a general activity network called EON (Event Operations Network) that will be presented in next chapter.

This chapter starts with an overview of the generic aspects of the data model in terms of hierarchy. On the subsequent sections, a more detailed view of the different object-oriented models used to characterize all the information needed in the scheduling model is shown.

3.2 Hierarchical framework overview

The work done on this thesis is based on a hierarchical organization of the information, which can be described as PSO (Process Stage Operation) framework.

At the upper level (Process), the material balance is performed in terms of the storable intermediates and the processes (and related recipes) leading to them (or consuming them). At the lowest level (Operation), the structure of activities involved in each process recipe

(tasks and subtasks) may be entirely described in terms of equality and inequality constraints for the starting, waiting and finishing times for each of such activities, which may be fully characterized after specifying their related general utility requirements.

The hierarchical approach presented allows the consideration of material states (subject to material balances and precedence constraints) and temporal states (subject to time constraints) at different levels. Other approaches such as the STN (Kondili et al., 1988) are based on states subject to both time and material constraints, thus being necessary the consideration of a huge number of states. However, this extreme situation is not excluded in this approach since it is possible to consider as many material states as temporal states.

Hence, this approach provides a flexible modeling environment to describe scheduling problems. Explicit material balance constraints may be specified for as many processes as needed or desired while implicit material balance constraints are defined within each process in terms of maximum and minimum batch size allowed. For each process any recipe may be specified in terms of equality and inequality constraints for the starting, finishing and waiting times of all the activities involved.

3.2.1 Process level

The Processing Network is defined using a network of processes and attached materials that describes the precedence for those substances considered. An explicit material balance is specified for each one of the processes in terms of a stoichiometric-like equation. Different processes leading to the same intermediate or final product may be specified for different material balances due to different recipes or equipment efficiency. Thus, this network provides a general description for production structures including common intermediates, synthesis and separation processes and recycling materials.

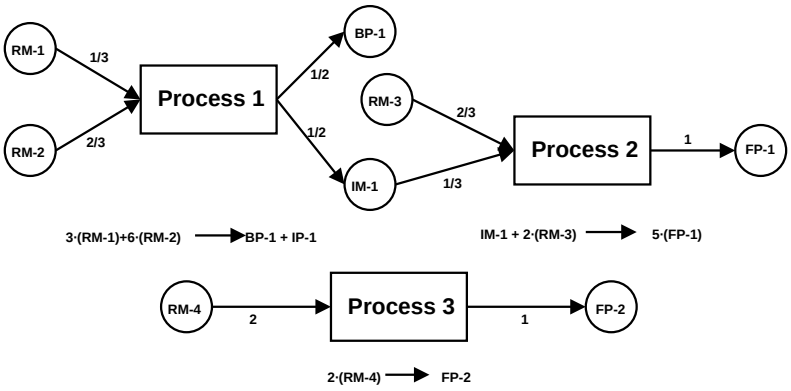

Figure 3.1: Processing Network for three processes leading to two final products and one by-product

This process network description fits perfectly the scheduling description in terms of a series of production runs (jobs or mini-jobs) associated to each of the processes. Following

the example in figure 3.1, the production scheme is made up of three processes leading to two final products (FP-1 and FP-2) and includes one intermediate product (IP-1). Each process may represent any kind of structure of activities necessary to transform the input materials into the output materials.

The feasibility of the schedules described in such a way may be easily checked by monitoring the accumulated levels of the materials involved in the sequence as shown by Graells et al. (1995, 1996). At this point, it is important to note the trade-off between complexity and flexibility offered by this modeling approach. While the consideration of too few processes may be an easy but rigid approach unable to describe useful situations, the consideration of too many processes may unnecessarily increase the complexity of the scheduling problem. Thus, the number of processes to be considered (which operations may be packed together and which should not) becomes a decision to be made based on practical experience.

3.2.2 Stage level

Between process level and the accurate description of the structure of activities described at the operation level there is a (auxiliary/supplementary/accessory/extra) but useful level which is the stage level. Following the classical definition, a stage is defined as the ordered set of operations to be executed in the same equipment unit. Hence, at the stage level each process is split into a set of stages **(**fig 3.2). Each stage will be defined as a sequence of operations so that by stage grouping some implicit constraints are defined:

- The sequence of operations defines a set of timing constraints (links) for the structure of activities to be defined later. This sequence of operations must be executed one immediately after another.
- Unit assignment is defined at stage level. Thus, for all the operations belonging to the same stage the same unit assignment must be made.
- A common size factor is defined for each stage. This size factor summarizes the contribution of all the operations of the stage.

Stage level is a useful concept for both the material balance (batch-size through size-factor) and the operation timing (implicit timing and assignment constraints).

3.2.3 Operation level

For each of the processes considered an activity network must be defined in order to fully describe the structure of operations involved. Following a finite wait policy (**FW**), each operation is defined by an operation time (TOP_n) and a limiting waiting time (TW_{max}). The simplest case corresponds to the single activity process, which just requires these two parameters to be specified. More complex recipes can also be defined declaring multiple operations explicitly linked in terms of constraints involving their starting (IT) and/or finishing (FT) times, thus building up a continuous time representation on a structured set of activities (Fig. 3.3).

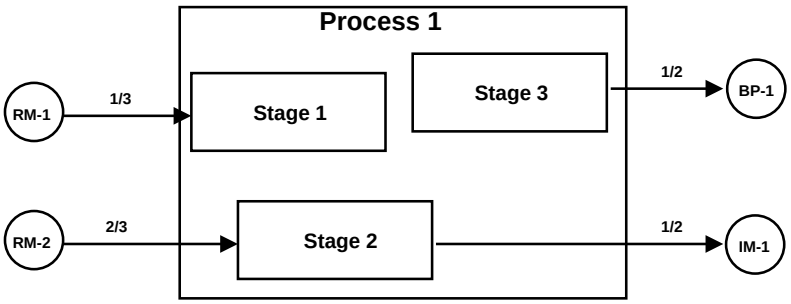

Figure 3.2: stage level

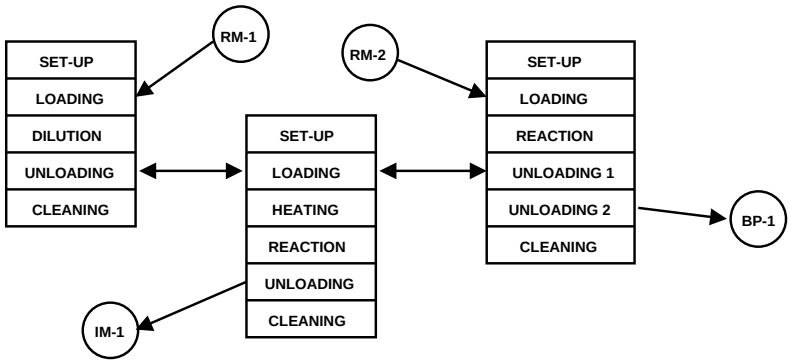

Figure 3.3: Operation level. Operation links are shown

While implicit time constraints (links) are given at the stage level, explicit time constraints (links) have to be defined at the operation level in order to fulfill the description of the structure of operations that define a process. This explicit links are shown in figure 3.3 by double direction arrows. However, the links between operations maybe of different kinds depending on if they express precedence / simultaneity etc. This point will be fully explained on section 3.4.2.1 at page 29.

Finally, general utility requirements (renewable, non-renewable, storage) may be defined at the operation level .

3.3 Resource Model

The main features of the resource model will be outlined in this section. Only the most important properties are shown in order to clarify the description of the data model.

Inside the resources classification the following aspects are included:

Materials: which can be either raw materials, intermediate products, by-products or final

products. In fact a material is anything that can be included into a material balance.

Units: A unit is a piece or a set of pieces of equipment. Sometimes it may be interesting to use all the equipments of a production train as an unique unit. This aspect may simplify the plant modeling from the scheduling point of view.

Profiles: A "profile" is any resource that needs an accurate tracking along the time in order, for example, to check that its production requirement does not exceed a maximum value. Examples of profile resources are: electricity, steam consumption and manpower.

Storages: A storage is a resource that can store one or more different materials within certain constrains of capacity and use.

An UML model of the resource structure is represented in figure 3.4.

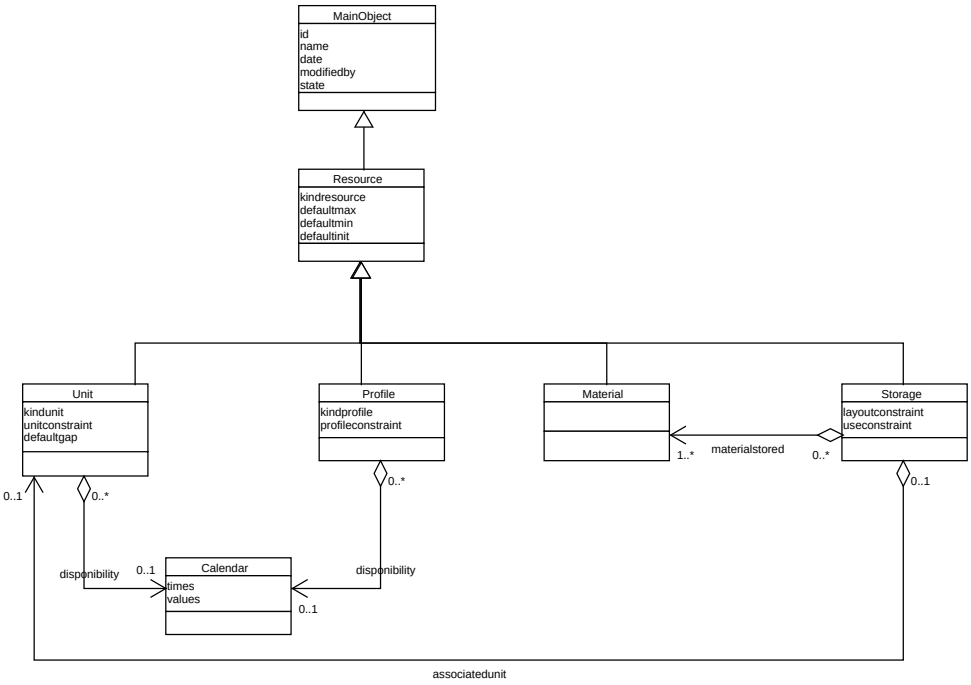

Figure 3.4: UML model of resources

As it it seen in figure 3.4 the parent class of resources (*Resource*) inherits of the generic class *MainObject*. The *Resource* stores information about the kind of resource and the default maximum, minimum and nominal to be applied

There are four defined kinds of resource: Units, Profiles, Materials and Storages. This field opens the possibility to add additional kinds of resources in a future.

The default maximum level can be used either to indicate the maximum capacity of a unit, the maximum available level of a profile resource such as electricity or the maximum storage level of a material in a specific storage unit.

The default minimum level can be used either to indicate the minimum capacity of the resource in the same way of the default maximum level.

The default initial level can be used either to indicate the nominal capacity of an unit, the initial available level of a profile resource such as electricity or the nominal or initial storage level of a material.

This maximum, minimum and initial values will be applied later as a default value for the resource constraints applied to a schedule. These schedule constraints will be explained on section 3.5.2.5.

Resource is a virtual class that acts as a base or parent class of all the resources. A detailed description of the derived classes will be performed in the following sections.

3.3.1 Material

The *Material* class involves the description of all the interesting materials involved in the process modeled. There is no difference between raw materials, intermediates and final products.

Material inherits from *Resource* so it inherits the properties of *Resource* and *MainObject*.

A material object can be related to one or more *Storage* objects. This relation involves that this material can be stored in the selected storages.

3.3.2 Unit

The *Unit* class involve the description of all the units involved in the planning and scheduling of the process.

Unit inherits from *Resource* so it inherits the properties of *Resource* and *MainObject*. Additionally, as it is seen in figure 3.4 it has additional properties like which kind of unit it is, the default occupation constraint to be applied, the default gap and the availability, which are described next.

The kind of unit splits the units in two big groups:

Discontinuous Unit: A unit, like a batch reactor or a batch column

SemiContinous Unit: A unit, like a pump or a packing line.

The occupation constraint defines the occupation constraint associated to the unit this constraint could be:

Default: The default time constraint implies that only one operation can be carried in the unit at the same time.

None: This option implies that there can be multiple operations carried out in the unit at the same time.

The default gap defines the default time gap necessary between tasks performed in the same unit. This value can later be replaced by the gaps defined at the level of unit operation in the schedule.

The availability defines the existence of a calendar for the unit. The calendar can be used to indicate period of maintenance or programmed stops of the unit.

3.3.3 Shared resource profile

The *Profile* class involves the description of all the resources to be traced during the scheduling of the plant, as steam,electricity, manpower, etc. *Profile* inherits from *Resource* so it inherits the properties of *Resource* and *MainObject*. It also has extra properties as it is shown in figure 3.4. These properties specify the kind of profile (therefore the way used to calculate it), and the default constraint to be applied.

In this work there are available three kinds of profiles:

Renewable: Describes a resource that once used is available again. Examples of this kind of profile can be, steam, electricity, vacuum, human resources, etc

Non-Renewable: Describes a resource that once used is consumed, some external actions are required to make it available again. An example of this kind of profile can be raw materials available as discrete parts (i.e. nuts, packaging materials, etc). Al the consumption takes place at the same time producing a ladder-like profile.

Storage: Describes a non-renewable resource that it is consumed gradually along the time elapsed in the operation thus generating a smooth profile.

In this work there are available two kinds of default deoverlapping policy. These policies only apply to the renewable profiles.

None: No policy is applied

Deoverlap: Deoverlap policy is applied by default if the kind of profile is renewable. Deoverlapping strategy is described in further detail in section 4.4 at page 51.

3.3.4 Storage

The *Storage* class is introduced in the model in order to deal with complex storage constraints. As in the other types of resources previously described, *Storage* inherits the properties of *Resource* and *MainObject*. It also has own properties which define the material stored and the constraints associated to the storage as the layout and use constraints

The layout constraint depends on the physical connections between the storage and other units. Each constraint also requires a different checking of the level constraints. There are six kinds of layout constraints predefined:

None: No layout constraint is defined for the storage; no level constraint is applied. In this case the input and output of materials are not taken into account when the timing model is applied.

SIO: Single input output constraint. It means that at any instant, only a charge or a discharge to/from the storage can be performed. It forces a sequence constraint for all the charges/discharges.

SISO: Single input single output constraint. It means that a charge and a discharge can be performed simultaneously, but no more than one.

MIMO: Multiple input multiple output constraint. Basically multiple input and output material transfers can be performed simultaneously.

MISO: Multiple input single output constraint. Multiple charges can be performed simultaneously with only one discharge.

SIMO: Single input multiple output constraint. Only one charge can be performed at any instant of time, but can be simultaneous to many discharges.

Each storage can have capability to store different materials in the same storage resource. The available options are:

Dedicated: Only one specific material can be stored (i.e. a dedicated storage tank)

Shared: Different materials can be stored simultaneously (i.e. a warehouse)

SharedExclusive: Different materials can be stored but only one at the same time (i.e. a shared storage tank)

A storage resource should have at least one material associated. Different storages can have the same material associated.

A unit can also be associated to an storage, as a unit can eventually be used as a storage.

The detailed description of how the storage constraints are integrated in the global tool is described in section 4.5 at page 55.

3.4 Recipe model

The recipe model should include all the aspects previously described. An UML diagram of the recipe model is shown in figure 3.5. The model is basically composed of the aggregation of the three main levels identified in ISA SP88 (recipe, stage and operation) plus a set of auxiliary classes that provide additional information about the recipe structure.

3.4.1 Main objects

The main structure of the recipe model is composed by three classes: *Recipe*, *Stage* and *Operation*. These classes identify the levels described at the beginning of this chapter.

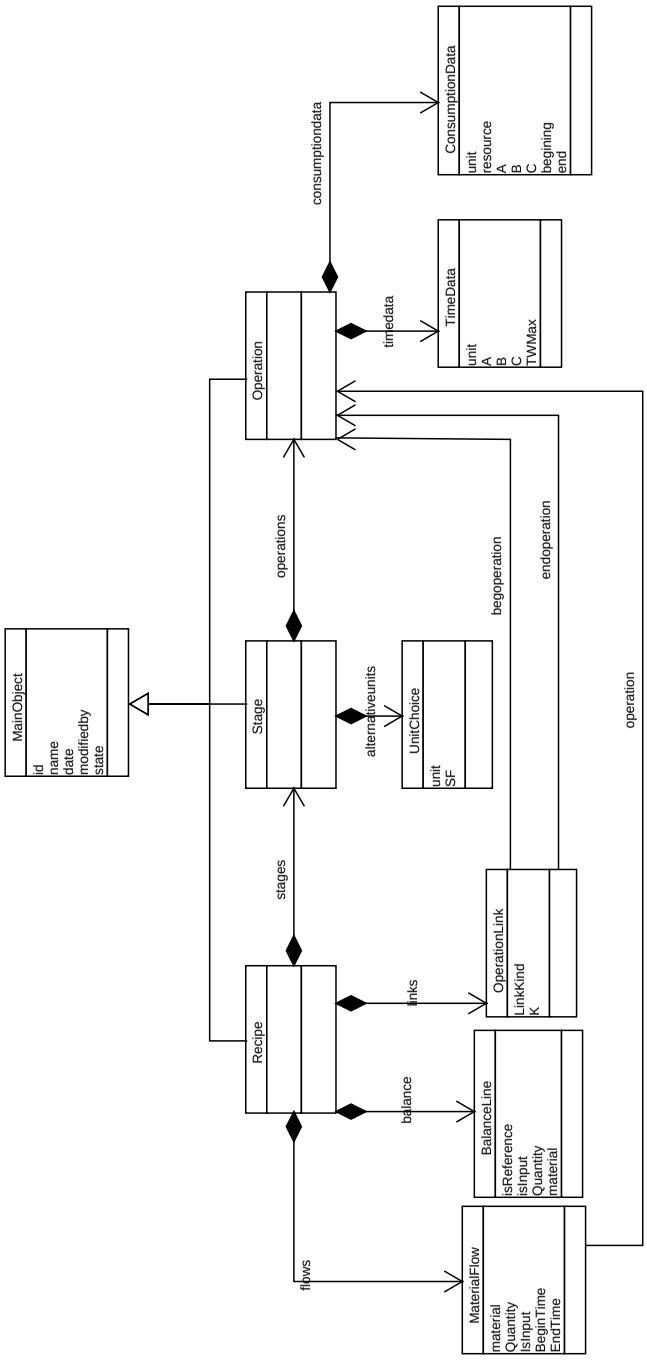

Figure 3.5: UML model of a Recipe and its associated objects

3.4.1.1 Recipe

The *Recipe* class contains information at the highest level of abstraction of the process description. The functionality is the same as the Master recipe level in the ISA S88 Standard for batch recipe description. *Recipe* inherits from *MainObject* and it has four properties that describe the material balance, the stages that belong to the recipe, the constraints between operations of the recipe and the time information about the material flows included in the recipe.

The first property is the balance, which is composed of array of *BalanceLine* elements, each of them containing the information related to the global material balances included in the recipe.

The global balance is not enough to describe in detail all the inputs and outputs of material, so the detailed temporal view of the balance is also defined. In order to define exactly when the inputs or outputs of the different materials are applied an array of *MaterialFlow* elements is used.

A *Recipe* class also contains the list of stages contemplated. It is an array of *Stage* objects.

Finally, the links are time constraints between operations inside a recipe. This property is an array of *OperationLink* elements.

3.4.1.2 Stage

The *Stage* class provides the recipe description with a detail level that follows the recipe level. As previously described in section 3.2.2, the stage contains an ordered set of operations and the list of available unit resources that can perform the stage. *Stage* also inherits the properties of *MainObject*.

Each *Stage* element contains an ordered set of *Operation* objects. Once scheduled the different *Operation* elements should be scheduled in the given order.

Additionally, an array of *UnitChoice* objects defines the units that can process the *Stage* and the size factor corresponding to the combination of *Unit* and *Stage*. The definition of the size factor is important because it allows the calculation of the maximum and minimum batch sizes for a given assignment of units to one *Recipe*.

3.4.1.3 Operation

As shown in section 3.2.3 (page 21), the *Operation* level contains the highest level of information detail included in this data model. The operation level contains the detailed description of the time and resources required for each operation. *Operation* inherits the properties of *MainObject* and it has the additional properties which defines the time and resources (other than the units) requirements.

The time requirements are defined as an array of *TimeData* objects, which describe the time requirements for the each *Operation*. These time requirements can be a function of the processing unit in which the operation is carried out.

The information about the resources (other than units) requirements is stored in an array of *ConsumptionData* objects. The consumption of resources may depend on the *Unit* resource used to perform the operation.

3.4.2 Auxiliary objects

3.4.2.1 OperationLink

The *OperationLink* objects is the way chosen to provide a description of the temporal constraints between operations within a recipe. A set of standard links has been defined in order to represent a broader view of the temporal constraints that can be found in the operation of process plants. The kinds of links available can be divided in two groups: the equality links shown in figure 3.6 and the inequality links shown in figure 3.7.

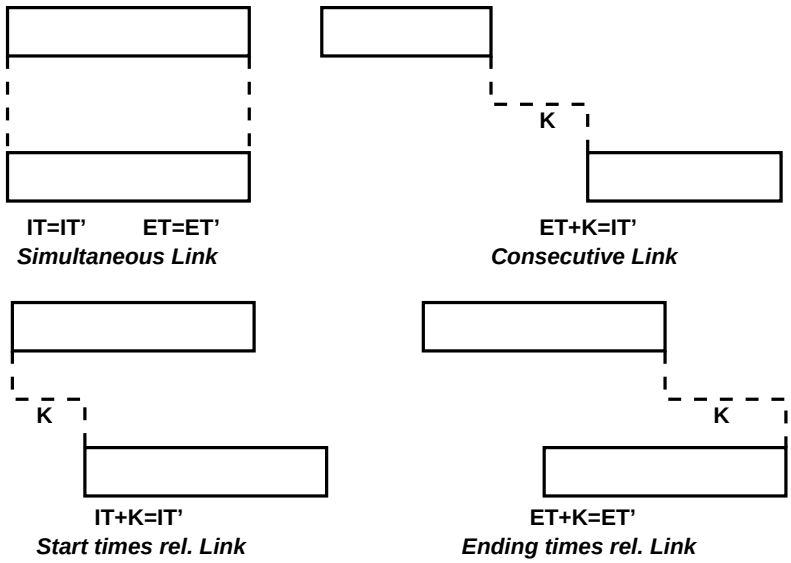

Figure 3.6: Kinds of available equality links

The equality links represent equality constrains between start and ending times of operations. As shown in figure 3.6 four equality links have been defined:

- **Simultaneous link**: This kind of link establishes that two operations should start and end at the same time. These constraints force that the operation time for both operations should be the same.
- **Consecutive link**: This link establishes a relationship between the end time of an operation and the start time of another operation.
- **Start times relationship link:** The start time of the second operation should be the start time of the first plus a delay. For example, this kind of link can stablish links like operation B should be started *k* minutes later that the start time of operation A.
- **End times relationship link:** This kind of link is similar to the previous one, but the times related are the ending times of both operations instead of the starting times.

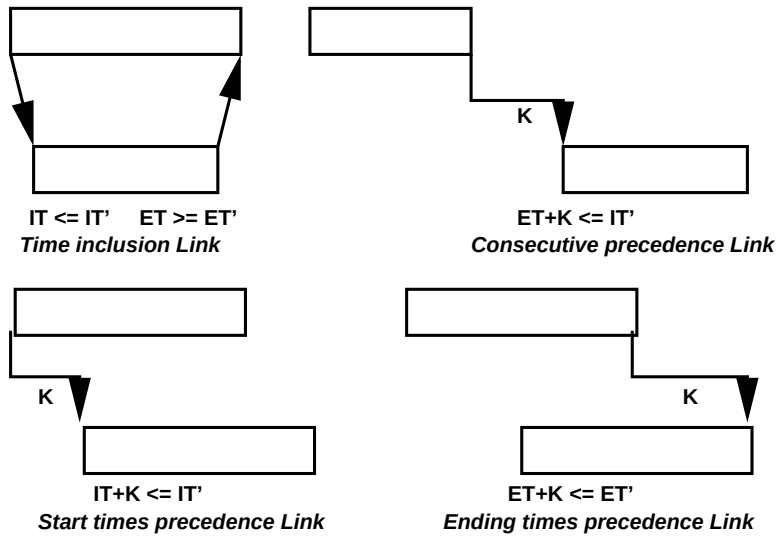

Figure 3.7: Kinds of available inequality links

In the same way as the equality links, the inequality links represent inequality constraints between the start and ending times of different operations. The pre-defined inequality links are shown in figure 3.7. In the same way as the equality links, four inequality links has been defined:

- **Time inclusion link**: This link can be used to define that one operation should be processed meanwhile another one is also processed.
- **Consecutive precedence link**: This link is used to define constraints like: operation B should be started later than *k* minutes after operation A is finished.
- **Start times precedence link**: This link is the inequality version of the start times relationship link. It is used when no exact relationship between the start times of both operations is known but a lower bound is required.
- **End times precedence link**: This link is very similar to the previous one. The times related are the ending times instead of the start times.

The OperationLink class has four properties to represent a link between two operations. These properties are: the first and second operations involved in the link, the kind of link to be applied and finally, the time gap (K) to be applied to the constraint:

This set of links allows to stablish complex time constraints within the operation of a recipe permitting the description of complex manufacturing systems. These links are directly translated into mathematical constraints when the schedule mathematical model is build. The detail of the mathematical constraints is shown later in section 4.3.2 on page 43.

3.4.2.2 BalanceLine

Each object of the *BalanceLine* class represents a line in the global material balance of the recipe. As show in figure 3.8, each balance line only shows the inputs and outputs of materials working with the whole recipe as a black box, that will allow to build a processing network as shown previously in section 3.2.1 (page20).

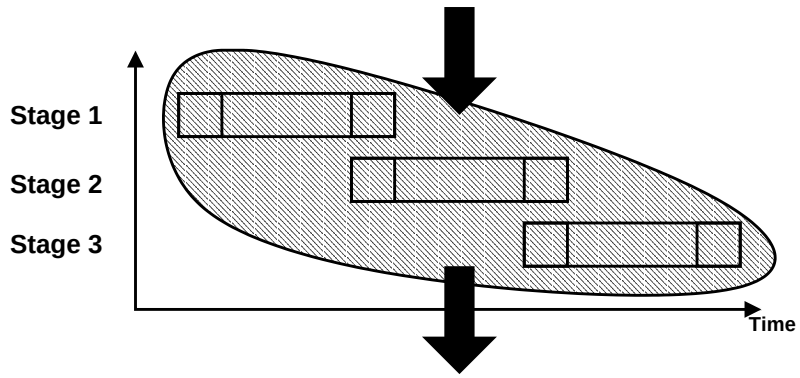

Figure 3.8: The balance lines only show the input/outputs of materials for the whole recipe.

The *BalanceLine* class has the following properties: the identification of the material that will be included in the recipe, the quantity of material in the balance, the specification if the material is an input or an output and finally, the definition or not of the material as a reference material (the material that will be used to express the batch size) . Only one of the materials involved in the recipe can be the *reference material*.

3.4.2.3 MaterialFlow

The *MaterialFlow* class is defined in order to provide a way to specify in which operations the inputs and outputs of materials are performed. As it is shown in figure 3.9 the idea is to describe exactly where the inputs and outputs of materials defined with the *BalanceLine* objects are carried on. The level of detail is higher than the one shown in figure 3.8.

The MaterialFlow objects have the following properties: The identification of the *Material* object involved in the flow, the identification of the *Operation* object involved in the flow of material, the definition of the fraction of the material involved in the flow refereed to the total amount of the material in the global balance (defined by the *BalanceLine* objects), the specification of the *Material* as an input or an output, the beggining time of the flow and finally, the end time of the flow.

The specification of the *Material* as an input or an output is included here because the *BalanceLine* objects describes the global balance, this is the global consumption and generation of the materials. But one material generated in the global balance can also be consumed

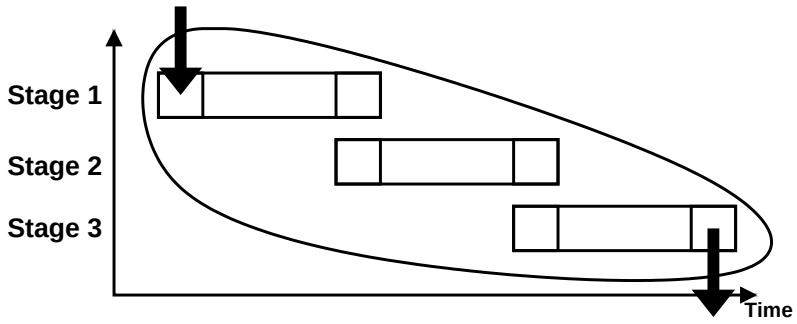

Figure 3.9: The Material Flows shows exactly where the inputs and outputs of materials are.

(i.e. due to recirculation). The addition of this property in the *MaterialFlow* object allows the description of these situations.

The beggining and ending times are expressed as a percentage of the operation following the time model shown in figure 3.10. a 0% is the beggining of the operation, 100 % is the end of the operation time (*TOP*) and also the beggining of the waiting time (*TW*). Finally 200% express the end of the waiting time.

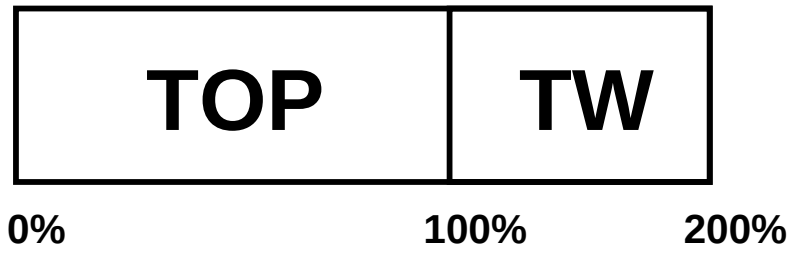

Figure 3.10: Time model of an operation

3.4.2.4 UnitChoice

An stage can typically be executed in different alternative units. The class *UnitChoice* is used to define the different alternative units to be used by an stage. *UnitChoice* defines a reference the *Unit* object and the associated size factor:

The Size factor can be defined as : $SF = \frac{unit\ capacity\ used\ in\ stage}{batch\ size}$. The size factor provides a way to calculate the maximum, and minimum batch size for a given assignment as it relates the unit capacity with the batch size.

3.4.2.5 TimeData

The *TimeData* class is used to specify the time requirements for an *Operation* object depending on the *Unit* used to carry out the operation. The default time model used in the proposed approach calculates the operation time as:

$$TOP = A + B \cdot Q^C \tag{3.1}$$

Q is the quantity of material processed in the task ($SF \cdot batch\,size$) and A , B and C factors that can be identified by knowledge of the physical model involved in the operation, by experimentation on a pilot plant or by historical data available. The time model used is shown in figure 3.10. Additionally to A , B and C factors a maximum waiting time (*TWmax*) is also defined. If the *TimeData* depends on the unit used, a reference to an *Unit* object should also be defined.

3.4.2.6 ConsumptionData

ConsumptionData class is used to define how the resources are used within an operation depending on the unit that carry out the operation. The calculation of the resource used follows the same model used in the previous class. The total amount of resource required is a function of three factors: A, B and C and the quantity being processed in the stage. Hence, the properties which define a *ConsumptionData* object are: the reference to the *Unit* object in which the resource consumption is dependent, the reference to the *Resource* which is consumed, the A,B and C factors, and finally, the beggining and ending time on which the resource consumption is applicable.

The beggining and ending time are numbers between 0 an 200 % according with the definition shown in figure 3.10.

3.5 Schedule data model

The schedule model includes all the required information to describe and calculate an schedule as it is shown in figure 3.11. The model is basically composed by the aggregation of the three main levels of ISA SP88 for an schedule (batch, unit procedure and operation) plus a set of auxillliary classes which provide additional information to the schedule structure: constraints associated to the schedule, demands and paths.

3.5.1 Main Objects

The main objects involved in the Schedule model are the *Schedule*, the *Batch*es, the *UnitProcedure* and the *BatchOperation*. A *Schedule* contains a collection of *Batch*es, each *Batch* contains a collection of *UnitProcedures* and each *UnitProcedure* contains a collection of *BatchOperation*s. These classes represent the four main levels of the ISA S88 structure for schedules.

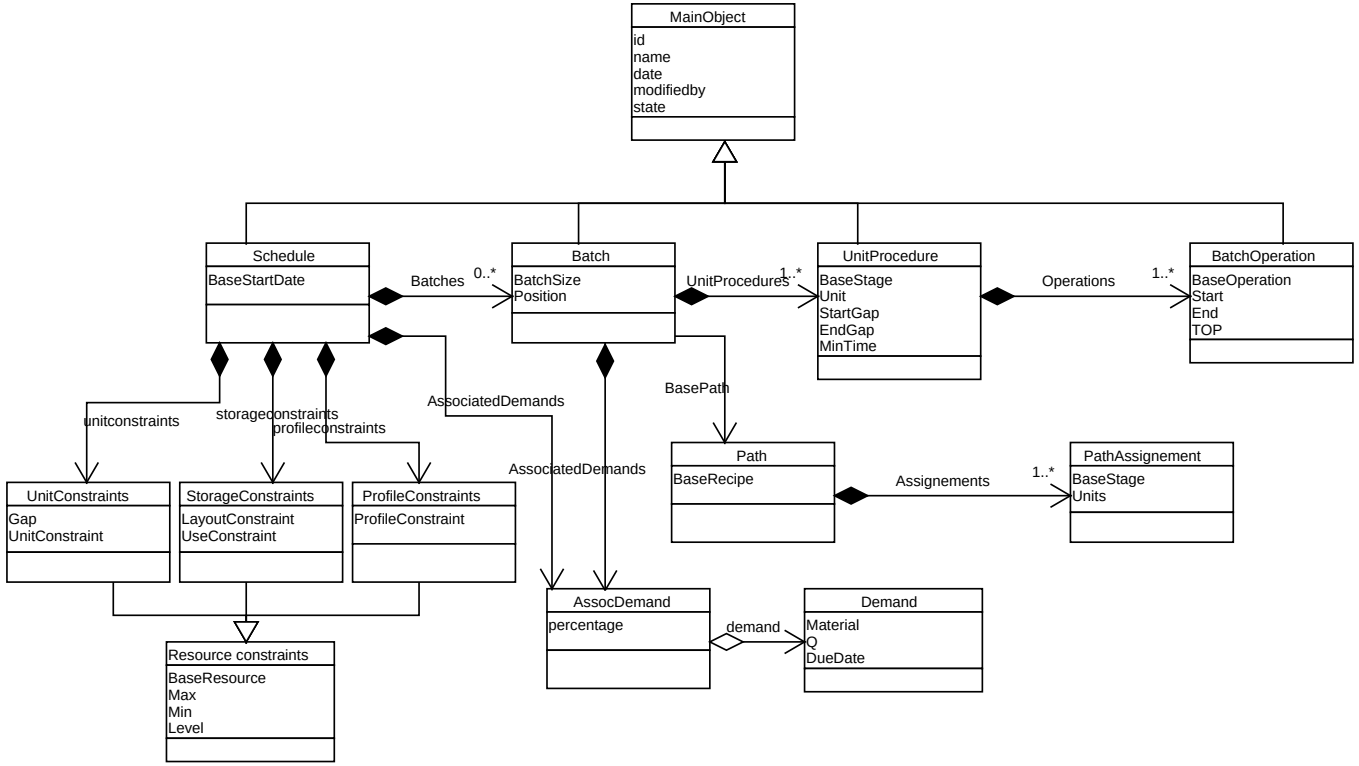

Figure 3.11: UML model of a Schedule and its associated objects

3.5.1.1 Schedule

The *Schedule* class is the container class for all the schedule model. It contains the list of batches and all the constraints associated to the schedule.

The schedule object contains the start date of the schedule. All the times of the operations included in the schedule will be referred to the start date of the schedule.

The Schedule contains also a list of associated demands that should be accomplished by the execution of the scheduled batches . A demand may only be partially associated to the schedule (i.e. because part of the demand is already fulfilled).

The constraints associated to the schedule specify what are the availability of the different resources used in the schedule as well as the specific constraints (i.e. storage layout constraints, unit sequence constraints, deoverlapping) to be applied in the specific schedule. These constraints have as default values those specified in the resource model, but they can be changed in the given schedule.

3.5.1.2 Batch

According to ISA S88, *Batch* is the entity that represents the production of a material at any point of the process. In fact, a batch represents the implementation of a master recipe into the plant. Each batch can be used to accomplish a part of a demand or a set of demands. A batch can be also be used to generate intermediate materials needed by other batches. The *Batch* class is used to deal with all the data related to an specific batch. *Batch* class inherits from *MainObject* and it has defined the batch size, the position, the list of associated demands, the base path and the list of unit procedures.

The batch size defines the quantity of the reference material (as defined in the used recipe) processed in the batch.

The position describes relation with the rest of the batches contained in the schedule.

The list of associated demands contains an array of *AssociatedDemand* objects which describes the demands that are partially or completely fulfilled by the batch execution.

The Path identifies the master recipe (a *Recipe* object), plus the specific assignment used in the batch (what *Unit* objects execute each *Stage*)

3.5.1.3 UnitProcedure

The *UnitProcedure* class is designed to manage the second level of detail in the batch description. Each *UnitProcedure* object contains all the *BatchOperation* objects that are carried out consecutively in the same unit. Each *UnitProcedure* may have also several time constraints associated, absolute or related to other unit procedures, as it is shown in figure 3.12. *UnitProcedure* class inherits the properties of *MainObject* and it has also additional attributes as: the reference to the *Unit* resource used, a reference to the *Stage* object executed in the *UnitProcedure*, the start and ending gaps as shown in figure 3.12, the minimum time and the ordered list of *BatchOperations* to be executed.

The start gap is the time constraint associated to the start time of the unit procedure and the end time of the previous unit procedure executed in the same unit (figure 3.12).

The end gap is a time constraint similar to the previous one, but relating the end time and the following unit procedure start time as illustrated in figure 3.12.

The minimum time is a lower bound for the start time of the unit procedure.

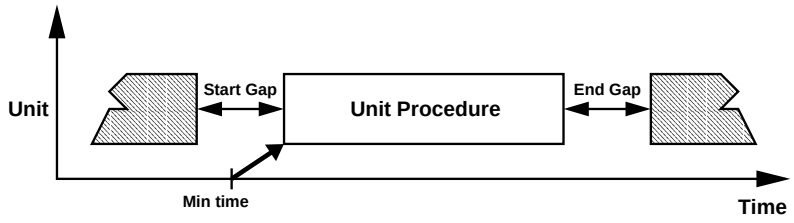

Figure 3.12: Constraints associated to an unit procedure

3.5.1.4 BatchOperation

The *BatchOperation* class contains the information related with the deepest level of detail of the batch execution related with the implementation of one of the operations described in the master recipe. At this level the calculated start and ending times are stored as well as the operation time and the base operation. A *BatchOperation* inherits form *MainObject* and has a reference to the *Operation* object that is executed in the *BatchOperation*, the start and ending times related to the beggining of the schedule and the operation time.

The Operation time is usually calculated using the equation 3.1. However this calculation can be discarded and the TOP could be fixed by the user.

3.5.2 Auxiliary objects

There are several auxiliary classes defined within the schedule model. These classes provide additional information about the assignment, the demands and the specific constraints applied to the calculation of the schedule.

3.5.2.1 Path

Path is the class which is capable to store specific unit assignments associated to a recipe. The main purpose is to store preferred assignments to be used in the plant. A *Path* object has a reference to a *Recipe* object which is being used to create the path and an array of *PathAssignement* objects that provides information about the specific assignment to be used for each *Stage*.

3.5.2.2 PathAssignement

This class is used to store a list of references to *Unit* objects that perform one *Stage* working on-phase. The *PathAssignement* class has also a reference to the *Stage* object being used.The *Unit* references should be consistent with the *alternativeunits* property of the *BaseStage*.

3.5.2.3 Demand

This class is used to store a demand requirement. This information is usually coming from an ERP system. The basic information associated to each demand is the due date (in which date the demand should be delivered) ,the quantity and, finally, a reference to the *Material* demanded.

3.5.2.4 AssocDemand

A demand may need more than one batch to be fulfilled. In that case, only a fraction of the demand is covered by a single bath. The *AssocDemand* class is used to register the fraction of the demand which is associated to a *Batch* or to an *Schedule*. Therefore, *AssocDemand* class has the following as properties the reference to the *Demand* object to be fulfilled and the fraction of the referenced *Demand*.

3.5.2.5 ResourceConstraint

As it is shown in section 3.3, each resource has a set of constrains associated. However an *Schedule* can be subjected to constraints which value differs from the value given in the *Resource* model. In order to be able to define the specific constraints applied to each *Schedule* object the *ResourceConstraint* class and its descendants are defined. The *ResourceConstraint* class provides the constraint information, which is common to all the resource constraints applied to one *Schedule* object. The values given by the Resource model are the initial values, but the user can change those values into specific constraints to be applied to an specific *Schedule* object.

3.5.2.6 UnitConstraint

The *UnitConstraint* class contains the same kind of constraint information about the *Unit* class shown in section 3.3.2 The difference is in that the *Unit* class stores the default constraint, whilst the class *UnitConstraint* stores the constraint information which is applied to an specific schedule. The *UnitConstraint* class inherits from *ResourceConstraint* and it has the following as additional properties the time default time gap necessary between tasks performed in the same unit, and the occupation constraint associated to the unit. This last property can have the values indicated in section 3.3.2 .

3.5.2.7 ProfileConstraint

The *ProfileConstraint* class contains the constraint information related to profiles applied to and specific schedule. The information found in this class have been explained earlier in section 3.3.3. This class also inherits from ResourceConstraint and it has the following as an additional property the information about the specific profile constraint associated to the schedule.

3.5.2.8 StorageConstraint

This class contains the storage constraint information applied to an specific schedule. As in the previous class, the *StorageConstraint* class contains the same constraints defined in section 3.3.4, inherits from *ResourceConstraint* and it has as additional properties the layout and use constraints applied to the schedule for an specific *Storage* object.

The different values of this constraints have been described previously in section 3.3.4 on page 25.

3.6 Conclusions

The data model presented in this chapter allows an organized and systematized information management dealing with the detailed representation of batch processes in the chemical industry. It allows the representation of:

- Recipes with an unlimited number of stages and operations.
- Time relationships between operations.
- Complex storage constraints
- Unit constraints.
- Material constraints.
- Calendar availability.
- Demand requirements.

The model is also extensible as it is object-oriented and allows the inclusion of new properties in all the defined objects. In the next chapters it will be shown how all these constraints are applied in order to calculate an schedule.

Nomenclature

ET_n**:** Finishing time of operation *n*

ET'**:** Finishing time of operation'

IT_n**:** Starting time of operation *n*

IT'**:** Starting time of operation'

TOP_n**:** Operation time associated to the operation *n*

$TWmax_n$**:** Maximum waiting time for the operation *n*